下载APP

15 | 分布式正确性的存在性（上）：什么情况下不存在分布式共识算法?

2021-01-27 任杰

分布式金融架构课 进入课程 >

讲述：任杰
时长 27:19 大小 25.03M

你好，我是任杰。这一讲我们聊一聊，什么情况下不存在正确的分布式共识算法。

对于金融行业来说，系统的正确性要远高于系统的执行效率。打个比方，当你在网上和朋友聊天的时候，漏掉了一两条消息其实无所谓。但是如果你给朋友网上转钱，钱转丢了就是件大事了。

金融行业的信息系统和互联网企业一样，也是由很多台机器组成的集群提供服务。机器一旦多了就会出现分布式系统常见的各种问题，比如宕机、网络中断。

那这种情况下，我们怎么才能保证金融系统的正确性呢？套用一句知乎上经常看到的评论，我们在回答为什么之前，先要问问是不是。你有没有想过，万一正确的分布式系统并

不存在呢？

这些问题其实是对分布式系统的深入思考，也是金融级软件对架构师的要求。只有知其然并且知其所以然了，客户才能对你做出来的金融系统有信心。

核心思路 & 小结

共识是指多台机器之间达成统一的结论。这节课我们会证明可能是分布式系统里最重要的一个结论，那就是不存在共识算法。准确来说，**在一个完全异步的分布式系统里，如果至少有一台机器可能会出问题，那么就不存在非随机的共识算法。**

从结论可以看出，想要共识算法不存在，需要同时存在两个现象：一个是机器出问题，另一个是完全异步。只要我们能让任何一个条件失效，就存在共识算法。

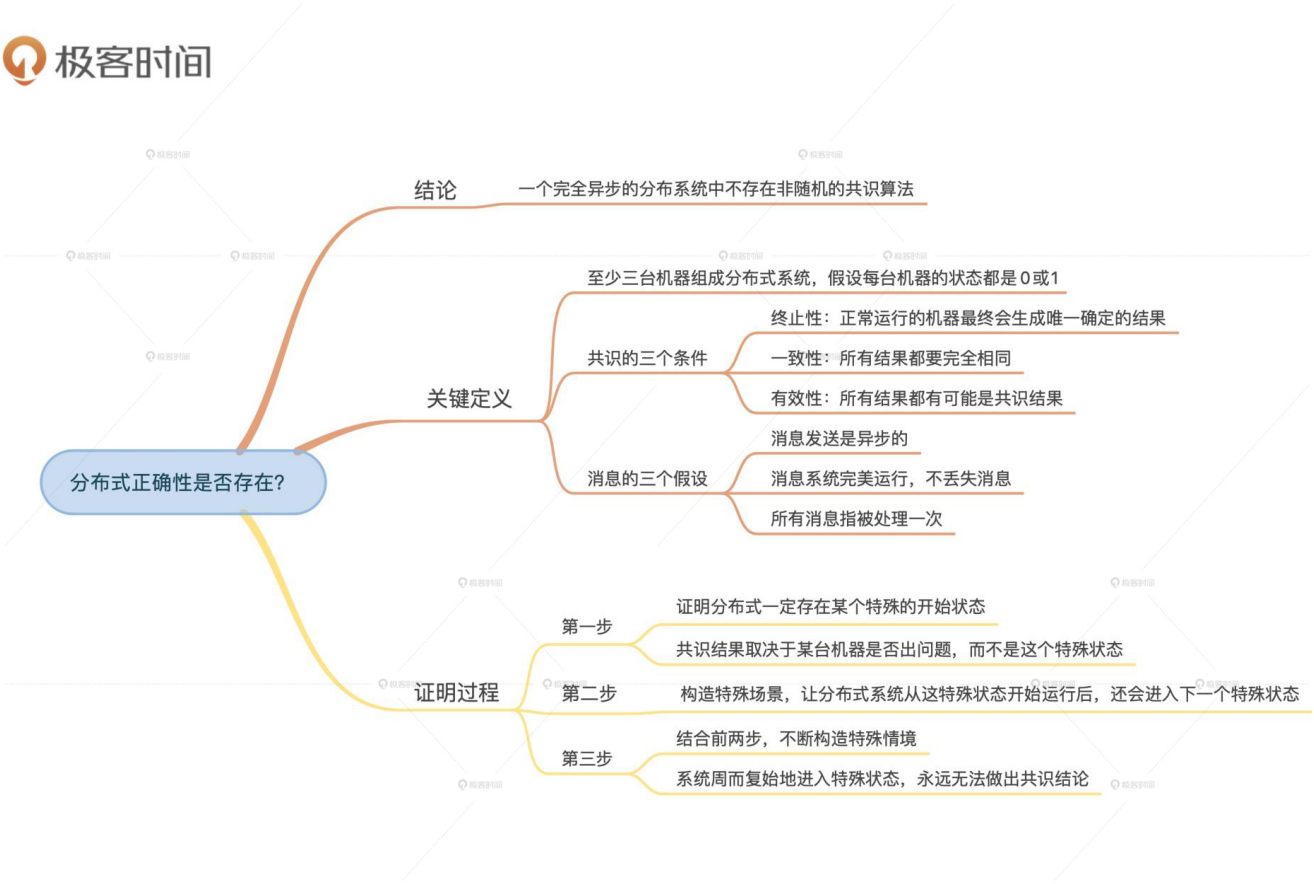
很显然我们无法保证机器不出问题，所以重点要放在怎么让系统不是完全异步。幸运的是**完全异步这个条件非常容易去除，只需要给每台机器增加一个时钟来判断发送的消息是否丢失。**

这也是为什么常见的共识算法，比如 Paxos 和 Raft，里面一定会判断消息是否超时。所以虽然理论上分布式系统不存在共识算法，但是现实中很容易绕开这些理论约束，实现正确的共识算法。

证明的过程分为三步。第一步证明，分布式系统一定存在某个特殊的开始状态，共识算法的最终结果与这个特殊状态无关，只取决于某台机器是否出问题。

第二步证明我们能构造出一些特殊场景，使得分布式系统从这个特殊状态开始运行后，还会进入下一个特殊状态。

第三步结合了前面两步的证明，我们不断构造特殊场景，系统会周而复始地进入特殊状态，从而永远无法做出共识结论。由于对于任何一个号称具有共识能力的算法，我们都可以构造出这些特殊的情境，也就不存在真正的共识算法了。



如果你想更深入地了解这个结论，可以看这几篇论文：

> 1985 年三位科学家 Fisher，Lynch 和 Patterson 的论文"Impossibility of Distributed Consensus with One Faulty Process"。这节课主要讲解的是这篇论文。
>
> 1996 年的论文"Unreliable failure detectors for reliable distributed systems"。这篇论文证明了如果分布式系统存在一个能让你最终做出准确判断的不准确时钟，那么系统存在共识算法。
>
> 1996 年的论文"The weakest failure detector for solving consensus"。这篇论文证明了这个不准确时钟是共识算法存在的充要条件。

好，现在我已经交代了这节课的核心内容，不过你要是有好奇心，可以仔细看看后面的证明过程。

算法课一般会讲如何实现一个算法，很少会谈到如何证明算法不存在。证明算法存在只需要举个例子就可以，但是证明算法不存在，就需要证明所有可能的算法都不行，难度非常高。你可以通过下面的证明，提高自己对分布式系统复杂度的理解。

背景及定义

背景

"Impossibility of Distributed Consensus with One Faulty Process"这篇划时代的论文在 2001 年被评为 Dijkstra 分布式系统最具影响力论文。因为这个结论太重要，所以一般称为“FLP 结论”。

这篇论文证明了分布式系统不存在共识算法。我们前面说过共识的定义，但不够准确。所以在证明开始之前，让我们重新定义一些专有名词。

定义

分布式系统和机器

分布式系统由**至少 3 台**机器组成。每台机器都有自己的**初始状态**。为了方便证明，我们假设状态是二元的，即只能是 0 或者 1。不是二元状态的情况和二元状态一样，只是证明过程会稍微繁琐一点。

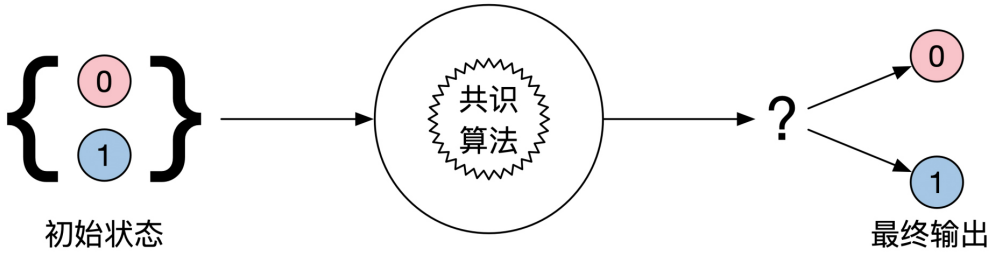
分布式系统是一个封闭的系统，没有外界输入。系统里的所有机器节点在最开始都有自己的状态，随后这些节点之间，会按照算法定好的逻辑给彼此发送消息。每台机器在收到消息之后可以做三件事。

第一，改变自己的状态。这时机器从初始状态变为中间状态。

第二是向其他机器发送消息。发送的机器数量不做限制，可以一个机器都不发，也可以发给很多机器。

第三是输出一个结果。这个结果只能从 0 或者 1 中选择，而且机器只能输出一次结果。

通俗地说，每台机器里面最开始含有一个数字，0 或者 1。机器最后会输出一个数字，也是 0 或者 1。机器之间可以互相发送消息，通过消息来改变彼此内部的状态。输入和输出示意图如下：



和现实世界类似，分布式系统内的机器并不稳定。它们会死机，会重启，也会变得很慢。但是我们这里假设机器逻辑都是正确的，一旦我们写了一个算法，机器会老老实实地按算法执行，不会偷偷利用算法的漏洞来攻击算法。

用现在的区块链理论来说，我们这里假设碰到的是非拜占庭问题，而不是区块链或者比特币解决的拜占庭问题。如果你对拜占庭问题有兴趣可以自行查找相关定义，或者参考这篇文章。

最后说说论文对共识算法的一个隐含假设。这里假设**共识算法不是随机的**，主要是考虑结果可复现性。

共识

有了前面对机器的准确定义，共识就更好定义了。共识其实就是要求这组机器的输出都相同，但是这还不够准确，更加精准一点的定义是这样的，共识需要满足以下 3 个条件：

1. 终止性（termination）

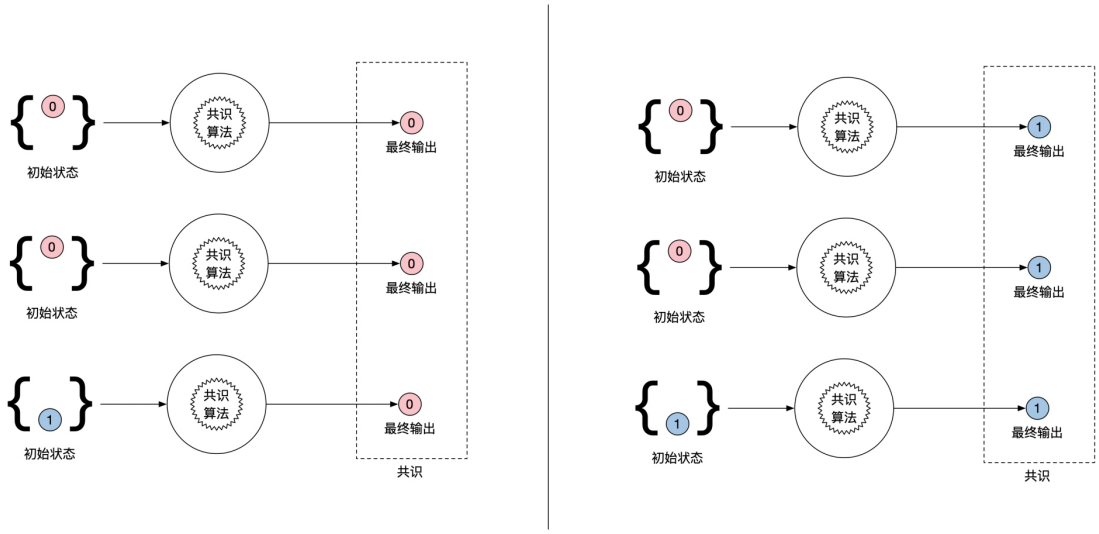
2. 一致性（agreement）

3. 有效性（validity）

其中，**终止性**指每个还能**正常运行**的机器**最终**都需要确定**唯一一个结果**。这里有两个重点。一是所有还能正常运行的机器都需要生成结果，一个不落。不能正常运转的机器没有任何要求。二是结论只能生成一次，一旦做出就无法更改。

一致性相对好理解一点。一致指所有结果都需要完全相同。**终止性**和**一致性**有一个更通俗的解释：所有机器的状态初始可以在 0 和 1 中任意选择，最后当共识算法结束的时候，所

有还活着的机器需要输出一样的结果，要么都是 0，要么都是 1。示意图如下：



最后一个条件是**有效性**。**有效性**指的是所有结果都有可能是共识结果。举个例子，你其实也可以参与分布式系统的决策。你可以写一个共识算法，不管机器初始状态是什么，最后都输出 0。这样也算是一个共识，但是这个共识算法毫无意义，所以我们需要把这种极端简化的情况排除。

消息

我们前面提到了机器之间是可以互相发消息的。消息需要仔细定义，因为分布式系统的复杂度是和消息传输方式的复杂度一一对应的。

消息的第一个假设是消息的**发送是异步**的。**异步**是指你给对面的机器发送了一条消息后，你不一定能收到反馈。

收不到反馈可能有很多种情况，这时候你不知道究竟是消息还没发送到那台机器，还是那台机器已经收到了，但是给你反馈前死机了。

收到消息的时间间隔也没有任何假设。你的消息可能几分钟后就能收到，或者很久才收到，甚至永远收不到。时间间隔是一个重点，我们在最后会再次提到。

消息的另一个假设是消息系统本身的**运行是完美**的。所有消息会被先存储在**消息队列**里。这个队列**不会丢失任何消息**。

另外，所有**消息只会被处理一次**，也就是说不会碰到消息重发的问题。一旦机器从消息队列里读取了一条消息，这条消息就会永远从消息队列里消失。这时候如果机器刚好又重启，那么这条消息就会永远在系统中消失。

虽然现实中不会存在这么完美的消息系统，但是由于我们解决的不是消息系统的问题，在这里对消息系统做完美假设，这有利于让你关注分布式系统本身。

消息的最后一个假设是消息的**接收是异步**的。消息的接收顺序是完全随机的，并不是先到先得。这意味着你给一台机器发送两条消息，先发的消息可能后到。

正在运行的机器会不断地从队列里拉取消息。当队列里没有和它相关的消息时，系统会返回一个**空消息**。机器也可以根据**空消息**的情况来改变自己的状态。

举例

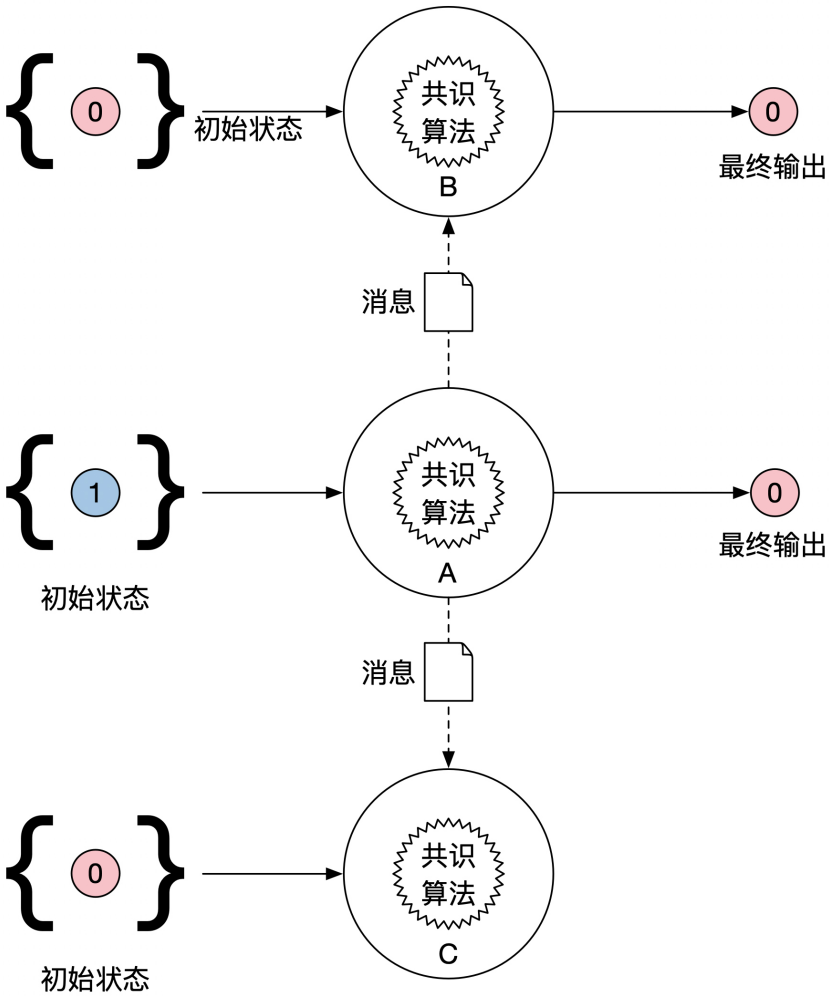
到这里为止我们明确了论文对分布式系统里机器、共识和消息这 3 个术语的定义。为了方便你理解，我在这里举个例子。

假设分布式系统由 A、B、C 三台机器组成，初始值分别为 1、0、0。系统在运行了一定时间之后，A 给 B 和 C 发了两个不同消息，同时 A 输出结果为 0。

B 过了一定的时间后，收到了 A 给自己的消息。B 这时候决定输出结果为 0，现在 A 和 B 达成了共识，都输出了 0。

C 收到了 A 给自己的消息后，没有给外面发任何消息，也没有输出任何结果。C 虽然一直在运行，但是没有输出任何结果，因此**不满足共识算法的终止性**。所以 A、B 和 C 这三个节点组成的分布式系统没能达成共识。

但是我们稍微改改对 C 的描述，结果就会大不一样。如果这时候 C 死机了，那么 C 就处于非运行状态。由于共识的定义对于非运行的机器没有任何要求，此时 A、B 和 C 组成的分布式系统就达成了共识。你可以再仔细体会一下两者的区别，下面是这个例子的示意图：



问题定义和解题思路

完成上面的定义后，我们终于可以对要证明的问题做一个准确概括了：**在一个完全异步的分布式系统里，如果至少有一台机器可能会出问题，那么不存在非随机的共识算法**。证明的过程分为三步。

第一步证明分布式集群存在一个特殊初始状态。我们无法通过这个初始状态预先知道共识结果，具体结果取决于哪些机器会在什么时候出问题。我们把集群的这个特殊初始状态称为**非确定状态**。

第二步，证明一旦存在一个非确定状态，系统在运行了一段时间后，一定还会进入下一个非确定状态。

第三步是将第一步、第二步合起来。从一个非确定性的初始状态开始，系统会运行到第二个非确定状态，然后会运行到下一个非确定状态，最后一直无限运行下去。这样就违反了

共识算法的终止性，也就证明了不存在共识算法。

证明过程用了反证法，比较精妙。接下来，你需要紧跟我的思路，仔细体会论证过程。

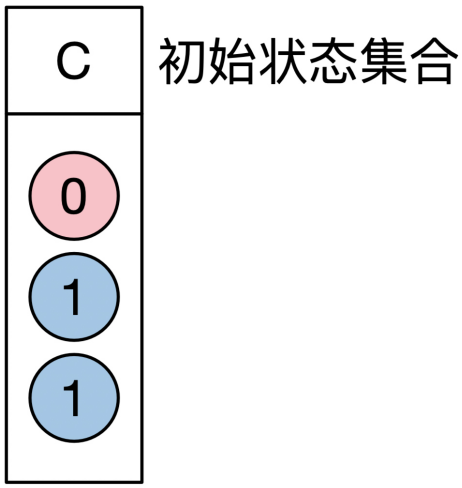
第一步证明

当分布式系统处于某种状态时，如果我们能提前计算出最后的共识结果，那么这个状态叫**确定性状态**。反之，如果最后的共识结果取决于机器是否在线，这个状态就叫**不确定性状态**。

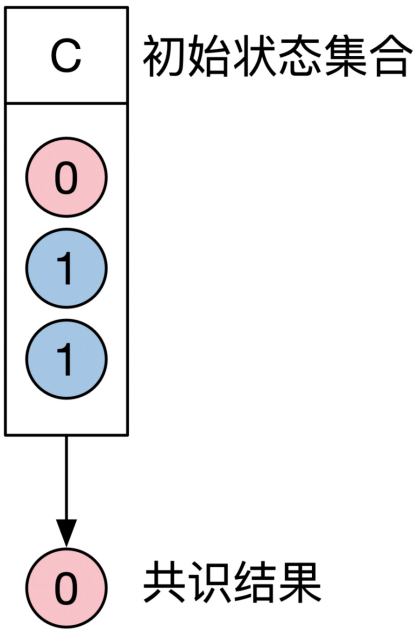
我们在这里需要证明的是，**任意一个集群都存在一个非确定性的初始状态**，即我们无法通过这个初始状态，判断最后的共识结果。

现在反证法开始了。反证法需要将需要证明的结论反过来描述，所以**我们假设从所有初始状态开始，不管机器是不是出问题，我们都能提前计算共识结果。**

我们先假设有一个初始状态的集合 C（configuration），C 包含了集群内所有机器的初始状态。比如下图画了 3 台机器和它们的初始状态集合 C：



根据反证法的假设，我们能提前计算出最后的共识结果，所以上图的初始状态也会有一个共识结果，比如说为 0：



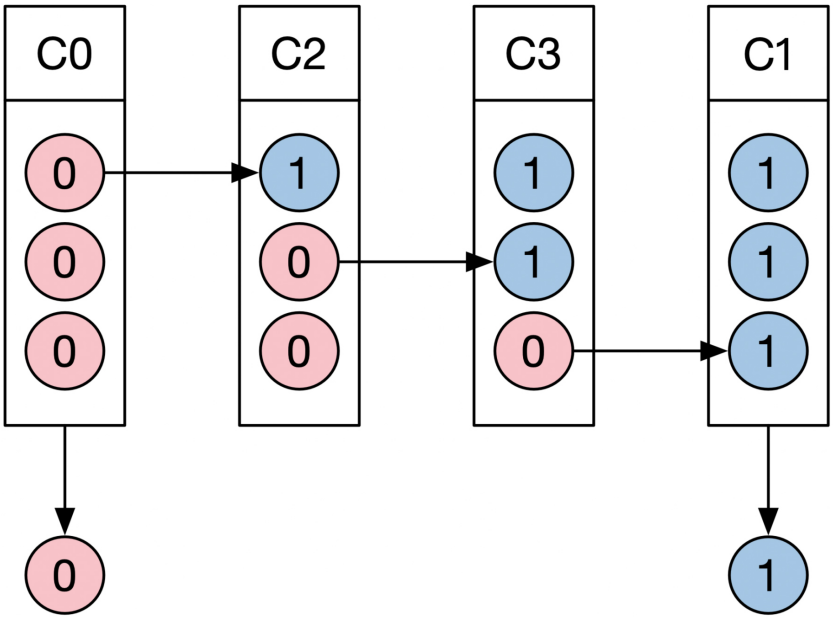
接下来我们需要用到共识算法的第 3 个特性：**有效性**。**有效性**表示当我们遍历所有初始状态时，一定有的初始状态最终会产生共识结果 0，也一定会产生共识结果 1。简单起见，假设初始状态集合为 C0 的时候，共识结果为 0。当初始状态为 C1 的时候，共识结果为 1。

下面就是反证法最精妙的地方了。前面提到的 C0 和 C1 是关于集群的初始状态。这两个初始状态虽然不一样，但我们可以一步步将它们变成一样的，过程很简单。

先选出 C0 里的一台机器，它在 C0 和 C1 的状态不一样。然后，将 C0 的所有状态复制一份到新建的初始状态 C2，并将 C2 里这台机器的状态变为它在 C1 里的状态。

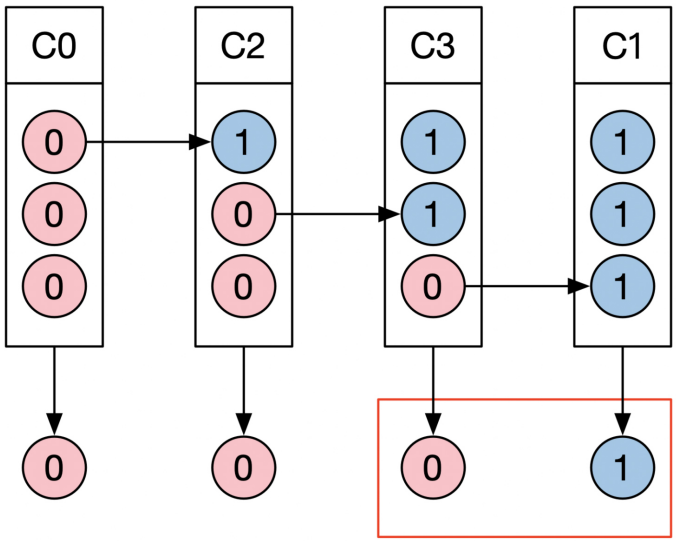
接着，在 C2 里找一台和 C0 状态不一样的机器，建立一个新的初始状态 C3，并将这台新机器的状态改变。以此类推，由于机器数目是有限的，最后一定会构建出一系列初始状态，它们之中只有一台机器的状态不一样。

举个例子，如下图所示，假设 C0 里的 3 台机器初始状态都是 0，C0 的共识结果为 0。C1 的所有初始状态都为 1，共识结果为 1。C2 将 C0 里的第 1 台机器的初始状态从 0 变为 1。C3 将 C2 里的第 2 台机器的状态变为 1。这样通过 3 次变化，我们最终可以将初始状态 C0 变为 C1：



那么问题来了。在上面这个例子里，C2 和 C3 所对应的共识结果应该是什么呢？其实不管它们对应的结果是什么，你会发现对于上面的 4 个初始状态，一定有相邻的 2 个初始状态，它们分别对应了 0 和 1 这两种不同的共识结果。你可以试试枚举所有共识结果的排列组合来验证。

我们假设 C2 和 C3 对应的共识结果都为 0，这样 C3 和 C1 这两个相邻的初始状态集合，就会有不同的共识结果：

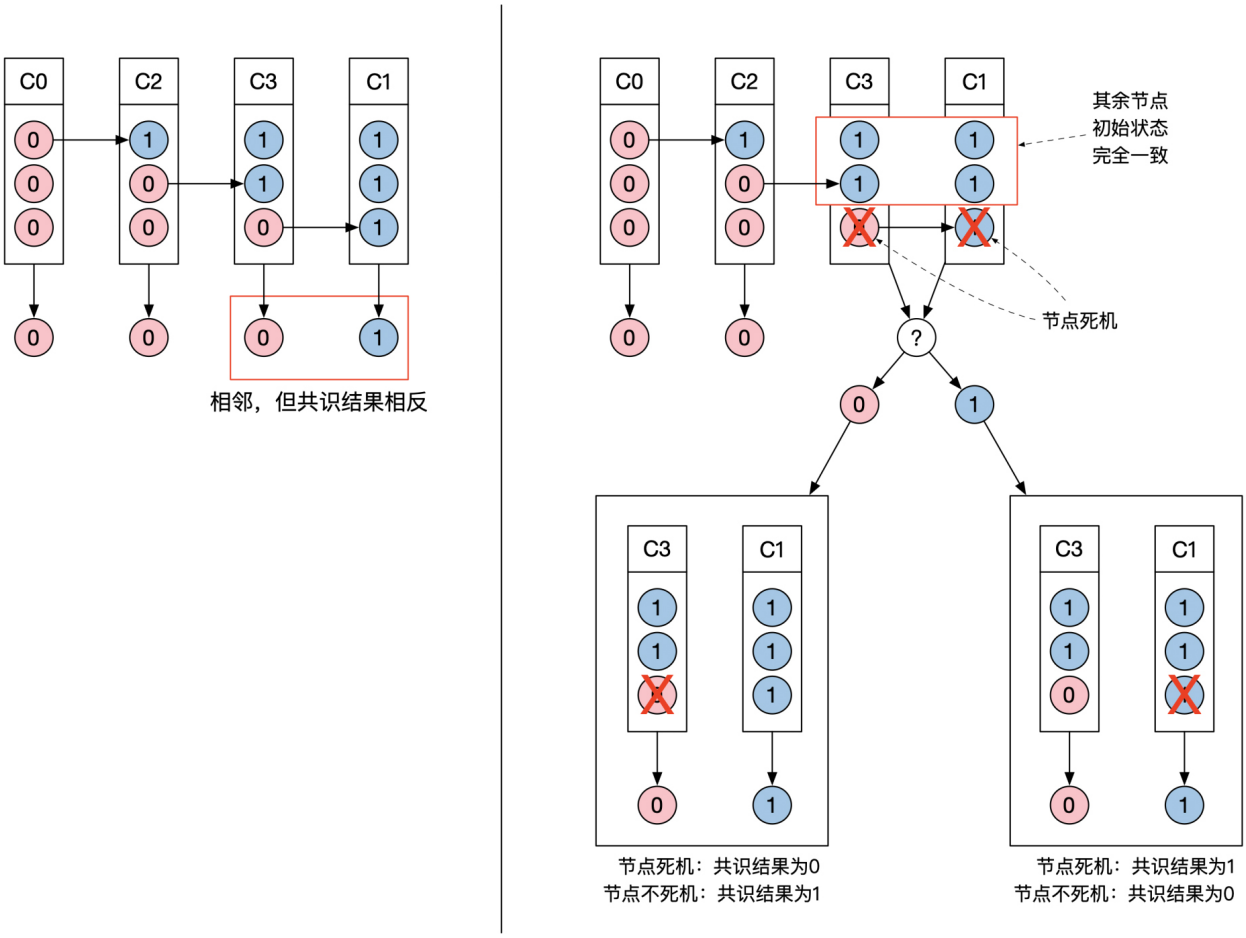


相邻，但共识结果相反

现在就到考验你的时候了。C3 和 C1 只有第 3 台机器的初始状态不一样。如果这第 3 台机器从一开始就死机了，C3 和 C1 的初始状态就会完全一样，这时它们俩都会产生怎样的共识结果呢？

下图展示了这个疑惑。反证法里假设过，集群的共识结果只和初始状态有关。那么如果第 3 台机器一直有问题，C1 和 C3 的初始状态其实是一样的，那么它们俩会产生一样的共识结果，要么是 0，要么是 1。

如果最后结果都是 0，那么 C1 在第 3 台机器不出问题时产生共识结果 1，但是当这台机器出问题后，会产生不同的共识结果，这和我们反证法假设矛盾。如果最后结果为 1，这样 C3 也会产生同样的矛盾。示意图如下：



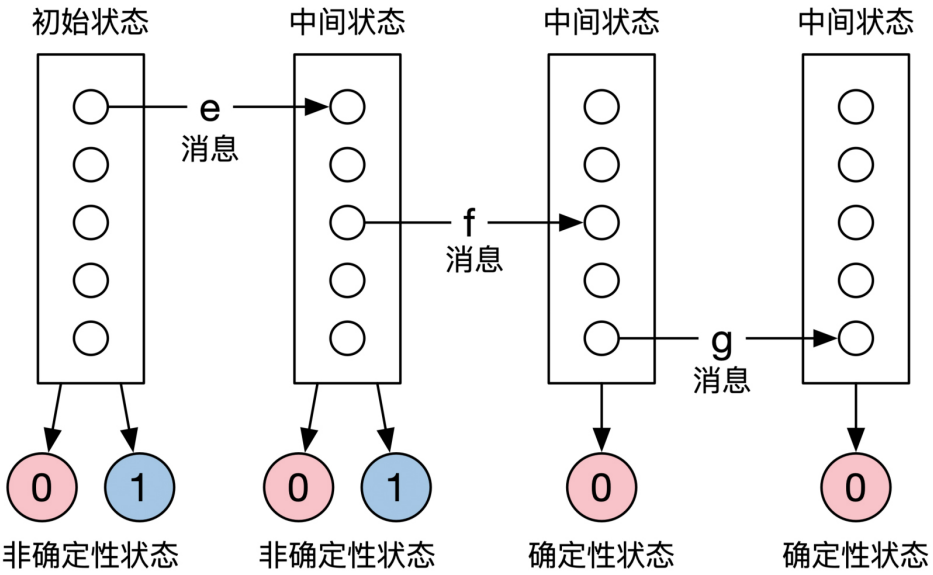
按照同样的道理，我们可以证明任意数目的集群都会产生类似的矛盾。所以对于任意一个机器集群，一定存在一个特殊初始状态，它的共识结果取决于一台特殊机器是否正常运行。第一步证明结束。

第二步证明

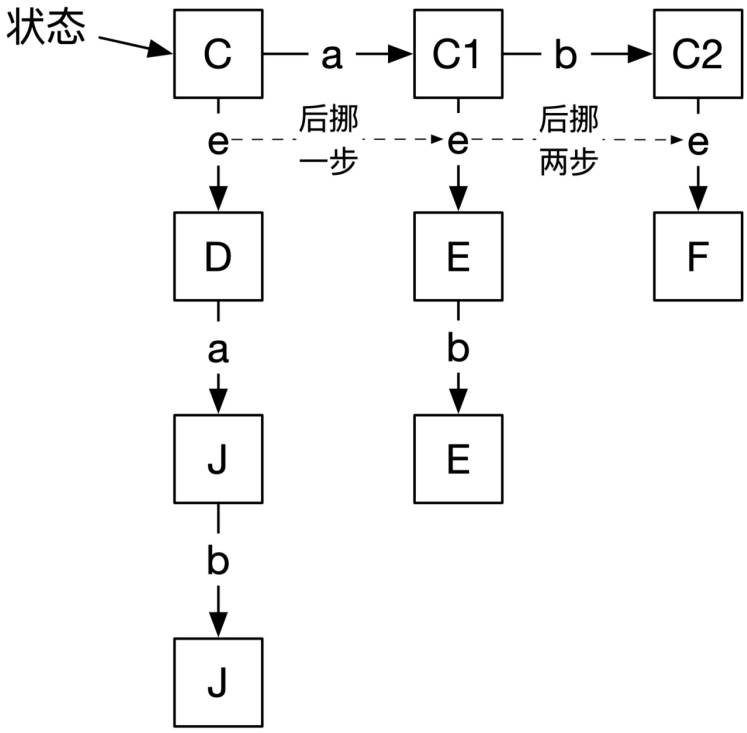
第一步证明只用到了分布式系统的**初始状态**和**最终结果**，而第二步证明则需要用到分布式系统的**中间状态**。和这一节课最开始类似，在证明之前我们再做一个定义。

分布式系统里消息的接收是有顺序的，尽管接收消息的时间差可能会很短，但是依然有顺序差别。所以，我们可以给分布式系统状态的变化定个顺序，任何两个相邻的状态变化之间是新接收到的消息，这个状态变化的顺序叫作**路径**。

和第一步证明一样，分布式系统的状态也分为非确定性状态和确定性状态两种。系统可能从非确定性状态运行至确定性状态，但是反过来不行。路径和两种状态的示意图如下：



这里还要说到一个新的操作。由于消息系统是异步的，消息的接收可以任意延迟。下图展示了对于某一条路径，将第一个消息 e 一步一步往后挪时，系统的不同运行状态：

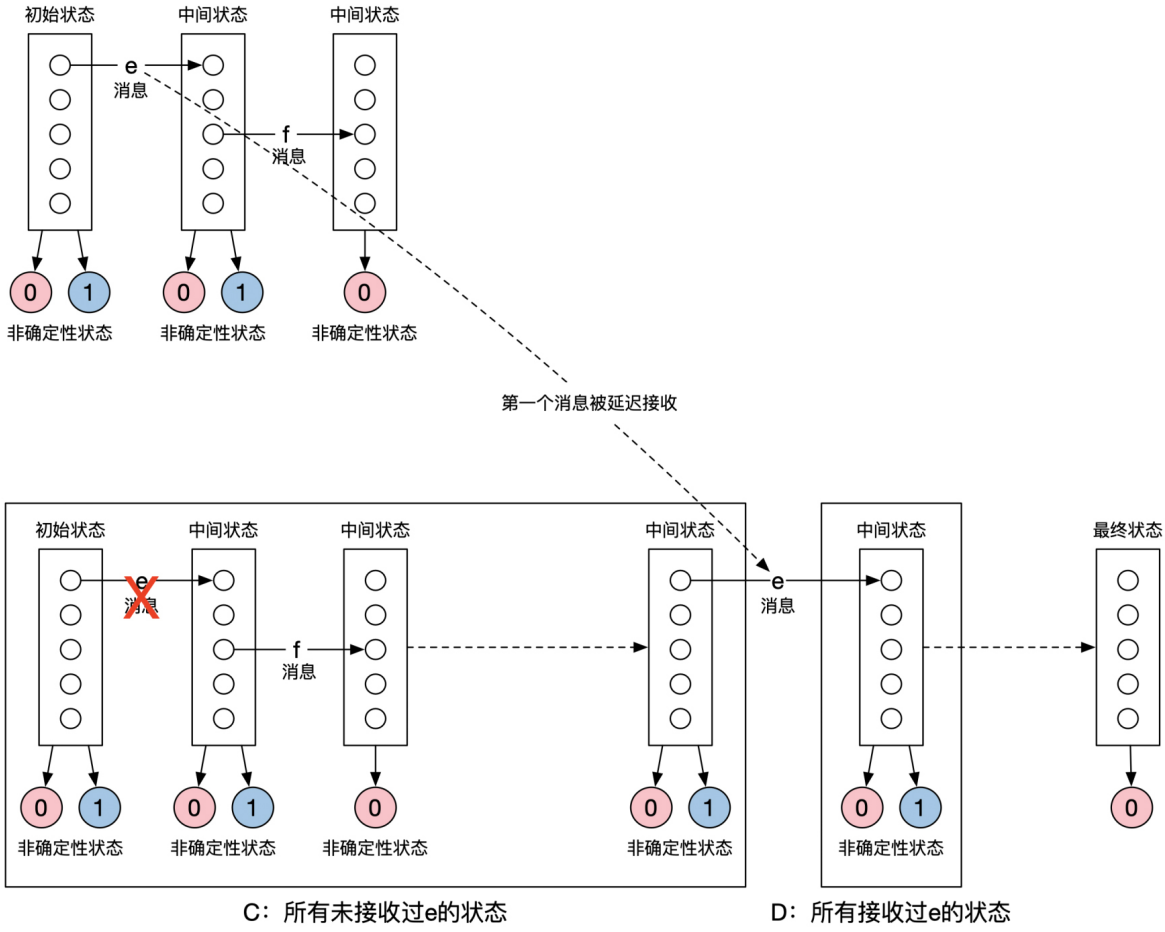


好，我们终于可以开始证明了。这里需要用到第一步的非确定性初始状态。

从这个状态开始，我们先随便选择一个消息，比如集群里可能会出现的任意一个消息 e。接下来，我们从集群所有可能的中间状态中，选择两大类出来。

一类是从来没接收过消息 e 的状态，我们称之为 C，另一类是刚刚接收过消息 e 的中间状态集合，称之为 D。剩下的中间状态跟证明无关，可以忽略。

那么，我们接下来证明，**在集合 D 里一定存在另一个非确定性的中间状态。**示意图如下：



这里还是用反证法，我们**假设集合 D 里所有的状态都是确定性状态。**

非确定性的初始状态一定会有两条不同的**路径**，分别产生 0 和 1 这两个共识结果。如果这个初始状态所有路径的最终共识结果都一样，那么就没有非确定性了。对于产生共识结果 0 的路径，如果这个路径没有穿过集合 D，那么表示路径在集合 C 里就结束了。

那么我们可以将消息 e 添加到这个路径的末尾。这样，构造出的新路径会穿过集合 D。由于路径在添加消息 e 之前共识结果为 0，那么添加消息 e 之后的共识结果也为 0。这里用到异步消息的一个属性：既然消息 e 出现过，那么消息 e 的接收时间可以任意调整。

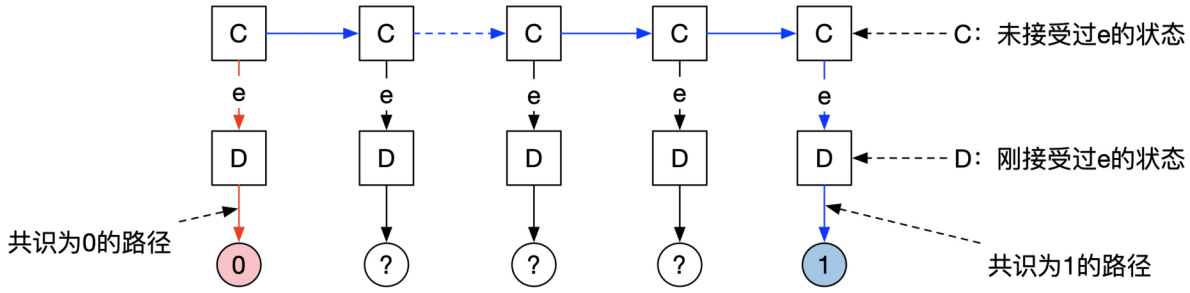
所以，一定有一条穿过集合 D 的路径会产生共识结果 0。同理，也有一条会产生共识结果 1。所以，集合 D 里所有状态不仅仅是确定性状态，它们一定能产生 0 和 1 两个共识结

果。

接下来选取两条路径。一条是第一个消息是 e 的路径，假设对应的共识结果为 0。由于初始状态是非确定性的，所以剩下的路径中，一定有一条产生不一样的共识结果。

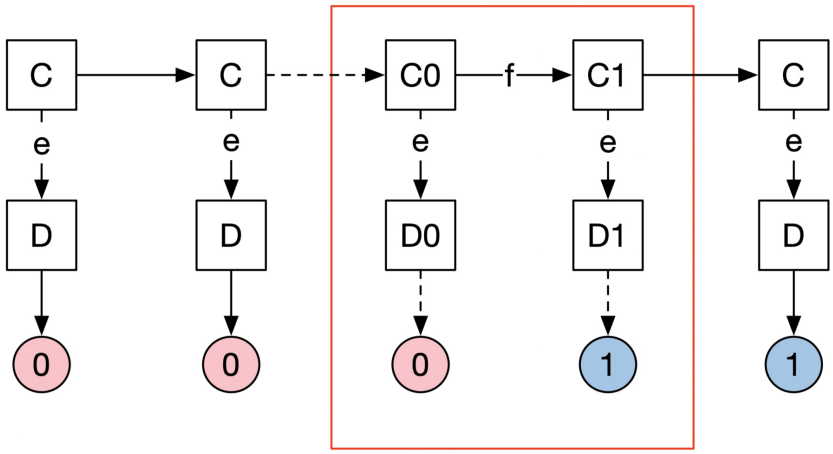
如下图所示，我们选取另一条产生共识结果为 1 的路径。如果这个路径不穿过集合 D，那么就可以按照上面的步骤，添加消息 e 到路径最后面，这样这条路径一定可以穿过集合 D。

然后，我们调整第一个消息 e 的接收时间，一步一步往后挪，这时候会产生一些新的穿过集合 D 的路径。那么对于中间的几个可能的路径，它们的共识结果是什么呢？



不管这些路径的共识结果是多少，和第一步证明类似，我们一定可以在集合 D 里找到两个相邻的路径，而它们的共识结果刚好相反。

如下图所示，我们假设状态 C0 在收到消息 e 后会进入状态 D0，D0 最终输出共识为 0。状态 C0 收到消息 f 后，会进入状态 C1，C1 收到消息 e 后，会进入状态 D1，D1 会输出共识结果为 1。

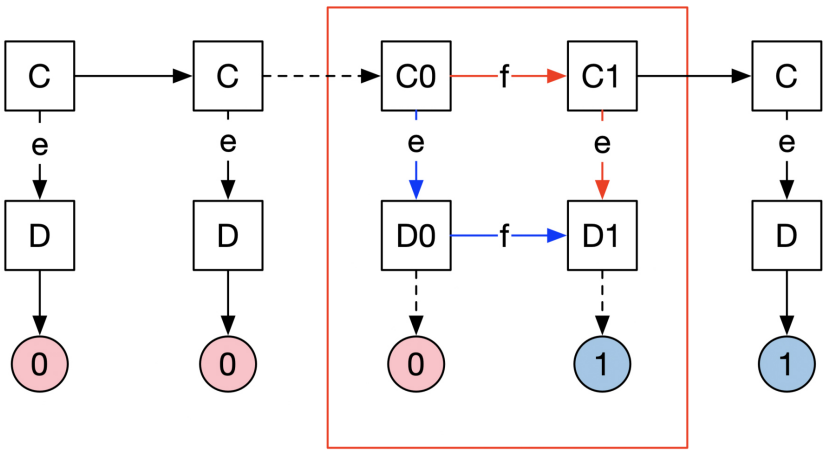


到这里为止，最关键的 4 个状态我们已经找到了。反证法的下一步是对上图消息 f 和 e 进行分析，看看**这两个消息的接收方**是否一样。

我们接下来会证明，**不管这两个消息的接收方是否一样，如果集合 D 的所有状态都是确定性状态，最终都会有逻辑矛盾。**

下面进入分情况讨论的环节。首先我们先看看**消息的接收方不一样是什么情况。**

这时候，我们将上图 D0 和 D1 之间增加一个消息 f，也就是把消息 f 的接收时间调整到消息 e 之后。这表示，我们有两条从 C0 状态到 D1 状态的路径。第一条是先接收消息 f，然后接收消息 e。第二条是先接收消息 e，然后接收消息 f。如下图所示：

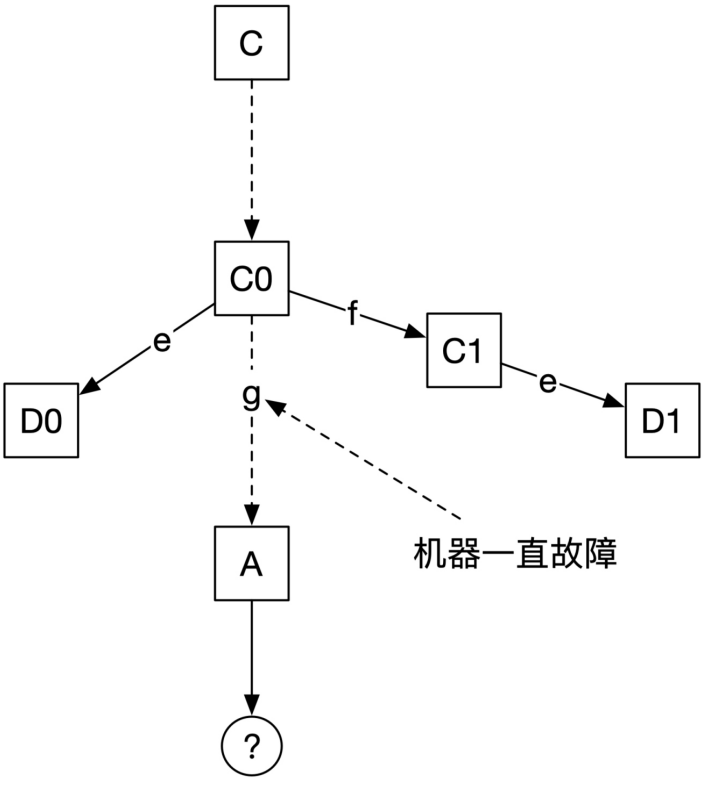


这时候出现了一个悖论。消息 e 和 f 对应了两台不同的机器，它们互不影响，所以 e 和 f 的接收顺序并不影响最后的共识结论。那么，上图从 C0 到 D1 的两条不同路径，最终应该导致同一个共识结果。这样看来，这个菱形的关系是正确的。

但是请你注意，我们在反证法里假设了状态 D0 是个确定性状态，它不能既产生共识结果 0，又在接收消息 f 后产生共识结果 1，所以这个菱形关系和我们之前的反证法假设矛盾。

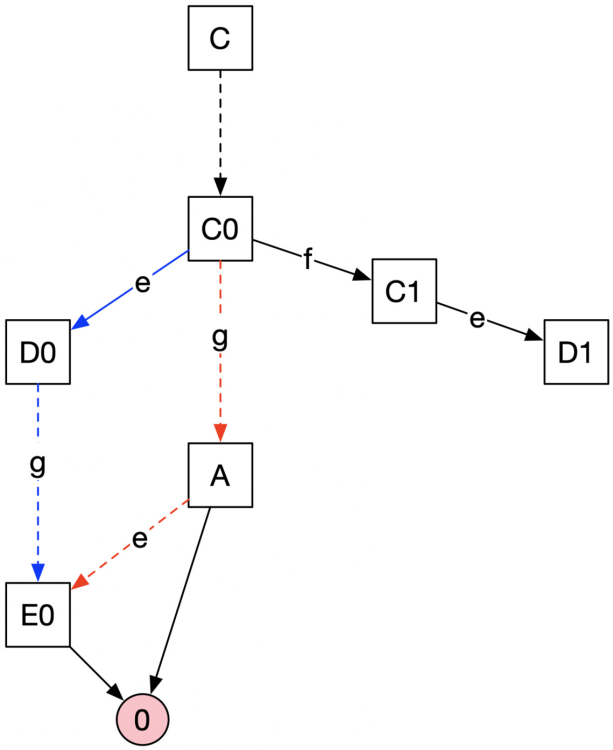
那我们再来看看**如果 e 和 f 的接收方一样，又会出现什么情况。**我们假设从状态 C0 开始，接收这两个消息的机器就出了故障，无法运行。系统在经过了一条路径 g 后，到达最终状态 A，并且产生共识结果。因为无限路径和共识算法的终止性相矛盾，所以路径 g 的步数是有限的，。

如下图所示，状态 A 究竟会产生什么样的共识结果呢？



先看上图的左边，我们构造两个场景。第一个场景是这样的。假设分布式系统从状态 C0 通过路径 g 到达状态 A 之后，原来一直出问题的机器突然恢复了。碰巧这时候消息 e 也刚刚到达，分布式系统会到达一个新的状态 E0。

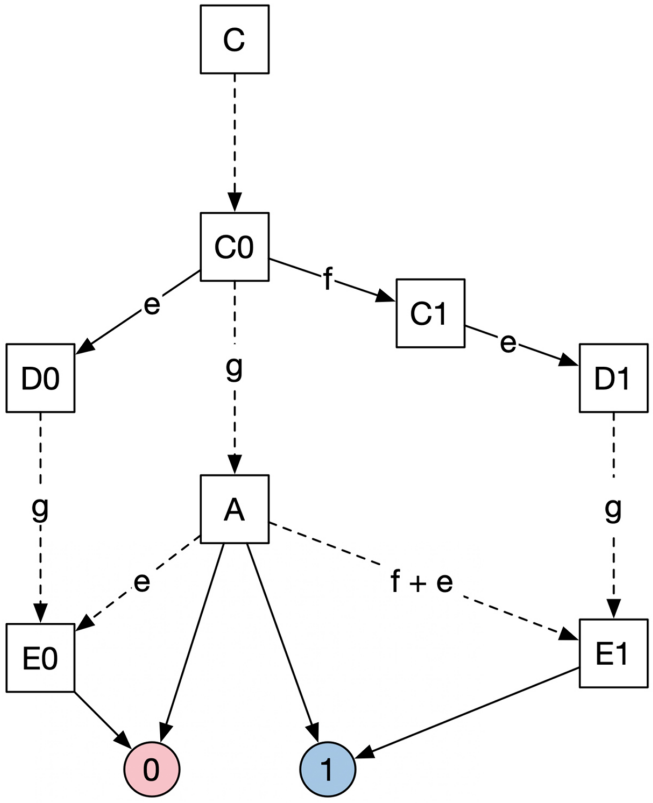
另一个场景从状态 D0 开始。由于消息是可以任意延时的，我们可以将路径 g 贴在状态 D 之后，这样状态 D0 在经过路径 g 后，也会到达一个状态。那么问题来了，如下图所示，这两个场景最终会达到同一个状态 E0 吗？



答案是会的。接收消息 e 的机器在路径 g 中一直无响应，所以消息 e 和路径 g 没有共同的机器，两者之间互换顺序不会影响最终结果。因此上图左边的菱形是合理的。两条不同的路径都会到达同一个状态 E0。

按照反证法的假设，D0 是一个确定性的状态，最终生成的共识为 0。所以 E0 最终会达到共识 0，这也意味着状态 A 也应该达到共识 0。

同理，我们也可以将右边用两条新的路径补全。一条是从状态 A 开始，添加一条 f+e 的路径。另一条是从状态 D1 开始，增加一条路径 g。这样右边也会达到同一个状态 E1。由于状态 D1 会达到共识结果 1，状态 E1 也会到达共识 1。下图画出了补全之后的情况：

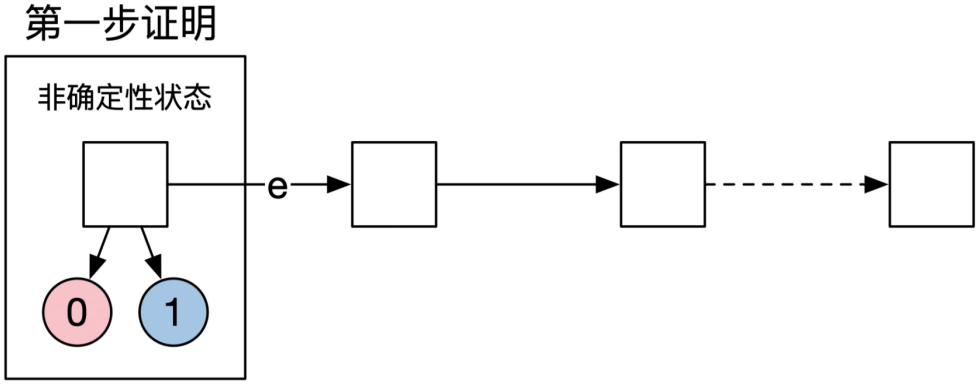


所以 A 这个状态既可以达到共识结果 0，也可以达到共识结果 1。这意味着 A 是一个非确定性状态，这和我们前面假设 A 是一个确定状态相矛盾。

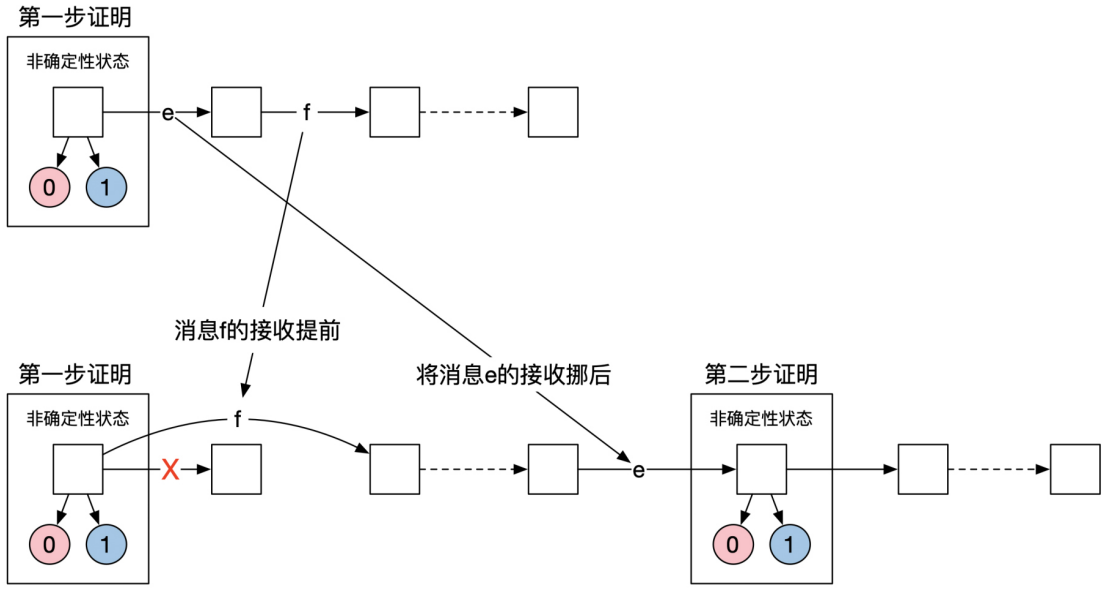
好了，到目前为止我们证明了，不管 e 和 f 这两个消息的接收方是否是一样的，如果集合 D 的所有状态都是确定性状态，最终都会有逻辑矛盾。所以反证法的假设不成立，也就是说，**集合 D 里一定存在非确定性的状态。**

第三步构造

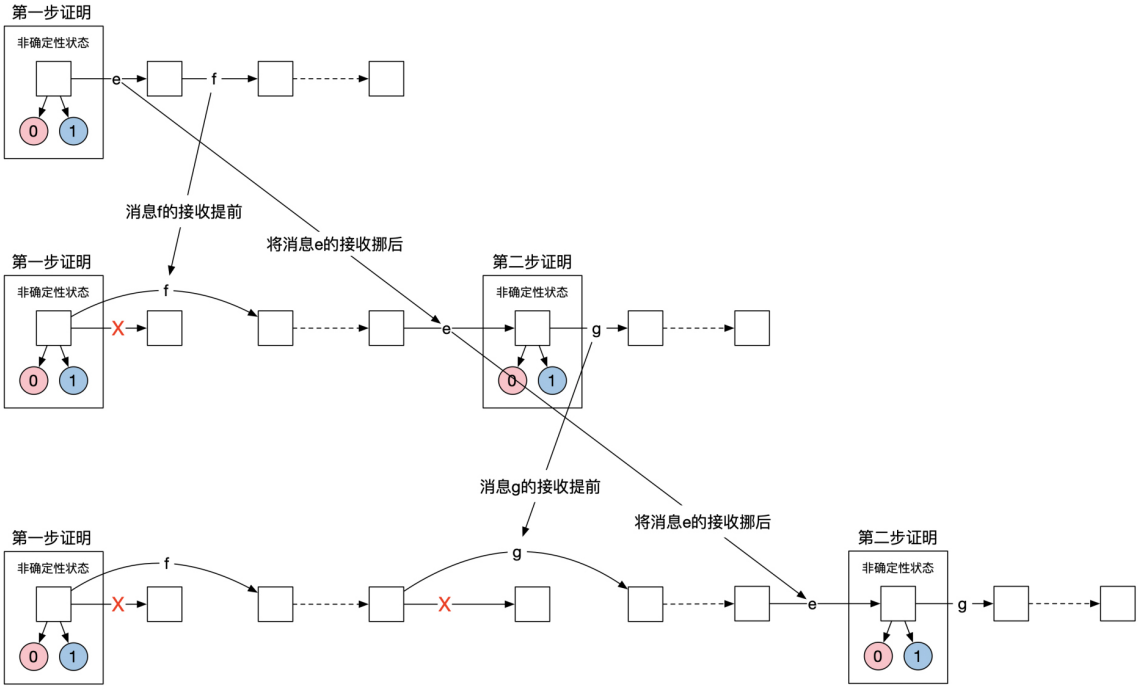
证明的第三步是构造出一个不会终止的共识过程，构造过程很简单。按照第一步证明，存在一个非确定性的初始状态，和它对应的第一个接收消息 e。如下图所示：



按照第二步的证明，我们能够找到下一个非确定性状态，这个状态刚刚接收了消息 e。如下图所示：



这样每当到达了一个不确定性状态，我们可以将消息 e 往后挪，从而制造出下一个不确定性状态。由于这个过程可以永远重复下去，系统会永远处于非确定性状态，这就违反了共识算法的第一个特性**终止性**。示意图如下：



思考题

1996 年的论文"Unreliable failure detectors for reliable distributed systems"证明了，如果在分布式系统里，存在一个能让你最终做出准确判断的不准确时钟，那么系统存在共识算法。这个时钟起到的作用是在分布式环境下，检测机器是否出问题。

失败检测分为两种属性：完整性和准确性。按照排列组合，一共有 4 种可能的情况：

1. 强完整性。所有正确的节点都会最终怀疑每个出错的节点。

2. 弱完整性。一些正确的节点都会最终怀疑每个出错的节点。

3. 强准确性。所有正确的节点都不会被怀疑出了问题。

4. 弱准确性。一些正确的节点不会被怀疑出了问题。

论文指出，就算只有很弱的失败检测，也能实现共识算法。你觉得这里的“弱”是指哪几种情况呢?

欢迎你在留言区跟我交流互动。如果学完这节课让你有收获的话，也欢迎你转发给同事、朋友，一起学习、探讨共识算法不存在的证明过程。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 14 | 正确性分级（下）：多机有容灾有哪几种不同的一致性?

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。